Keyword Search Relevancy Boosting

Goal

Provide a way to change the boosting used on fields in a keyword query.

Traceability

CMR-1335 - JIRA project doesn't exist or you don't have permission to view it.

Background

An effort has been undertaken to identify means of improving the relevancy of the search results returned by the CMR for keyword searches. Keyword searches are free text searches across multiple fields within the metadata. Unlike other types of searches in the CMR, results matching keyword searches are assigned relevancy scores based on the fields that match the given query. Each field is a assigned a "boost" value and the boosts of all the fields that match a given query are multiplied together (boosts are always >= 1.0) to generate a relevancy score. So the more fields that match, the higher the relevancy.

The boosts used by the CMR were chosen ad hoc based on perceived importance of the fields, so the "optimal" values for these boosts is not known. Adjusting these boosts to try to improve the relevancy was identified as "low hanging fruit" that could be easily implemented. Repeatedly changing the hard coded values and deploying to a separate instance for testing by third parties is cumbersome and slow. For these reasons we desire to make these boost values adjustable on a per-query basis.

Based on feedback in the comments section below this is not intended to be a documented API. It is provided for experimentation and client developers should not rely on its continued existence.

Requirements

1. Current boost values should be the default
    a. If no boosts are specified with a keyword query then the values currently used by the CMR should be employed.
2. Users should be able to specify any number of the boosting fields.
    a. If ANY field boosts are specified than any NOT specified will be set to 1.0 (no boosting for that field).
3. Attempts to set boosts on non-keyword queries should be rejected with a 400 error.
4. Attempting to set boosts for fields not used in keyword queries should be rejected with a 400 error.
    a. Currently boosted fields are
        i. short-name/long-name
        ii. project short-name/long-name
        iii. platform short-name/long-name
        iv. instrument short-name/long-name
        v. sensor short-name/long-name
        vi. spatial-keyword
        vii. science-keywords
        viii. temporal-keyword
    b. There are additional fields indexed for keyword searches. These should be boostable as well
        i. summary
        ii. version id
        iii. entry title
        iv. provider id
        v. version description
        vi. data center

API Options

There are several solutions that could work to solve this issue. The both involve adding parameters to the search API to specify the boosts:

1. Provide separate parameters for each boost.
    a. ..&short-name-boost=1.4&science-keyword-boost=1.8, etc.
2. Provide a 'boosts' parameter that works similar to the 'options' parameter
    a. ...&boosts[short-name]=1.4&boosts[science-keyword]=1.8

3. For JSON queries
    a. Use either of the parameter approaches given above
    b. Incorporate the boosts into the keyword query JSON
        i. {"condition": "keywords": {"values": "value1 value2" "boosts": {"short-name": 1.4, "sensor": 1.8}}}

In each case a default map of boost values would be created from the existing boosts and this map would then be updated with the boosts specified in the parameters/query.

Neither of the first two option is necessarily stronger than the other from a usability standpoint, but the second option allows for more concise implementation and easy extension to support new fields. For the third option incorporating the the boosts into the JSON query makes sense as they are part of the final elastic query, this could be left to a separate issue.

Recommendation

Option 2 provides an easier implementation path than option one for parameter queries, and it can also support JSON queries until a full JSON query solution can be implemented. For these reasons option 2 is preferred.

Meeting the Requirements

1. Current boost values should be the default:
    a. The default boost map would be based on the existing values in the CMR code.
2. Users should be able to specify any number of the boosting fields:
    a. Using the parameters a user can override one or more default values.
3. Attempts to set boosts on non-keyword queries should be rejected with a 400 error.
    a. Boosting for JSON keyword queries was added in Sprint 42 by means of consolidating the parameter and JSON keyword query approaches. Determining whether or not a query contains a keyword condition happens at the last step before constructing the native elasticsearch query. For this reason, this validation cannot occur during normal parameter validation, but must be deferred until the query is fully constructed. The upside is that this allows for validation to be consistent between parameter and JSON queries.
4. Attempting to set boosts for fields not used in keyword queries should be rejected with a 400 error:
    a. The same validation mechanism used for the 'options' parameter can be used to check the provided fields.

Modifications to the Query Model

A `boosts` field will be added to the query model to contain a map of fields to boost values. This field will be used to construct the elasticsearch query and to validate requirement 3.